Task 01: Basic Docker Virtualization (6 Marks)

* Create a dockerized environment in your computer/ PC (Linux environment preferred. Can be hosted inside a VM) and run at least 3 well-known docker images from the docker hub (i.e., hello-world, BusyBox, Nginx, Redis, Alpine,...etc.).
* Show the running docker containers in your machine. Remove all the running containers and their images and show the list of images in the virtual domain.

Table

Description automatically generatedText

Description automatically generatedRun three images: hello-world, BusyBox and alpine. Because these images are simple examples so it exits automatically after running. I use 'docker ps -a' to show all exited containers.

Text

Description automatically generatedDelete containers using command 'docker rm':

Text

Description automatically generatedShow docker images:

Delete images:

* Follow the instructions at https://www.docker.com/blog/how-to-use-the-official-nginx-docker- image/ to run a basic Nginx web server in your PC/VM. Add a custom HTML message with your Group members’ names and run the server. Show the outcomes.

Run the basic Nginx web server:

Graphical user interface, text

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

Text, letter

Description automatically generatedCreate a custom html page and run with -v flag to create a bind mount volume:

Graphical user interface, application, Word

Description automatically generated

* Boot up a basic ubuntu container. Install any Linux package inside the container, such as nano or IP-Utils.

Start container:

Text, letter

Description automatically generatedInstall nano:

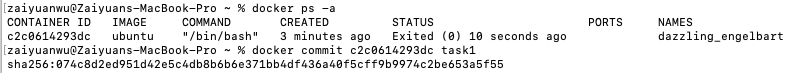
Text

Description automatically generated

* A picture containing graphical user interface

  Description automatically generatedCreate a directory with your group name inside the container. Inside the container, create a text file and include the group members’ names in it. Show the created directory and the file content.
* Commit the Ubuntu container and push its image to the docker hub. Mention the link to the exported docker hub image.

Graphical user interface, text

Description automatically generated

Link: <https://hub.docker.com/r/willinie/task1/tags>

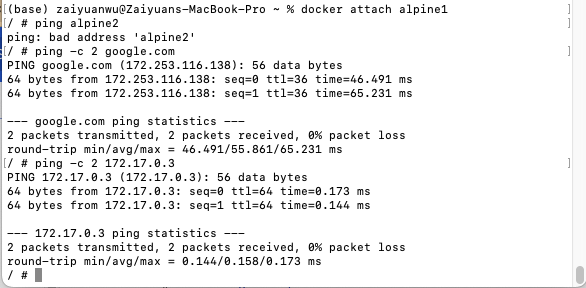
Task 02: Docker Networking (6 Marks)

https://docs.docker.com/network/

* Follow the instructions at https://docs.docker.com/network/network-tutorial-standalone/ use a default bridge to connect three alpine containers and run ping commands to test their connectivity. Show the ping results.

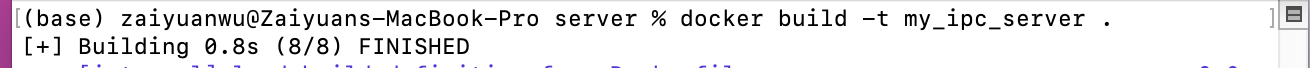
Text

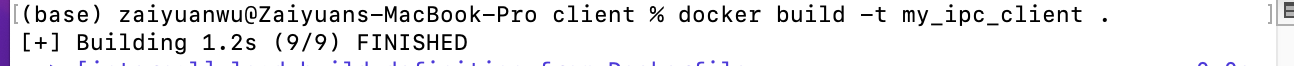
Description automatically generatedRun 3 alpine containers:

Test ping command:

* Establish an Inter-Process Communication (IPC) channel between two Ubuntu containers following the instructions at https://medium.com/techanic/docker-containers-ipc-using- sockets-part-1-2ee90885602c
* Leveraging the created IPC channel, send parametric values (at least 50) from container 1 to container 2, emulating an offloading scenario; compute the mean, median, and standard deviation of these parameters at container 2. You might have to extend the python script to establish the IPC communication. Send the computed stats to container 1 and display all the results at each container.

Build images:



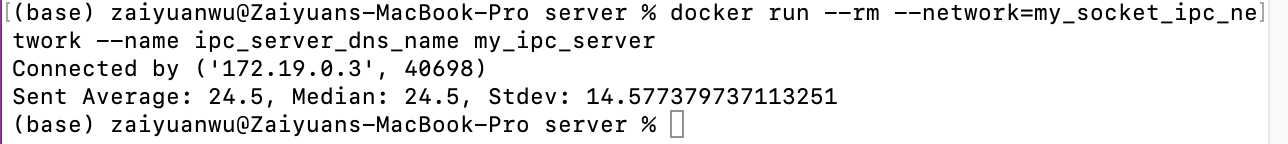


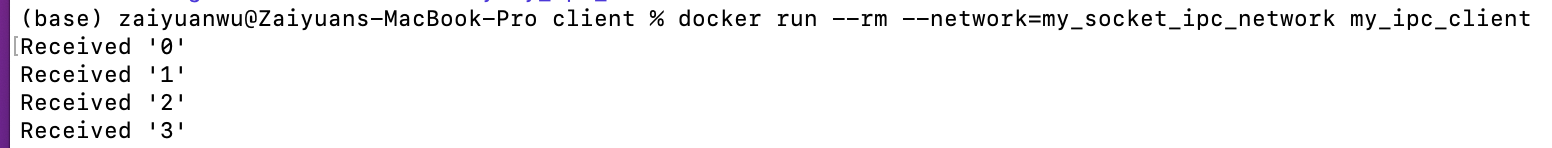
Define bridge network:

Table

Description automatically generated

Run containers separately:





Text

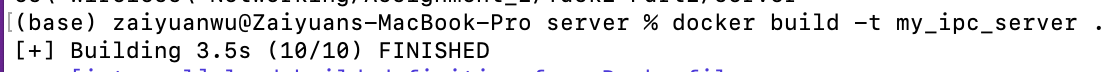
Description automatically generated with medium confidence

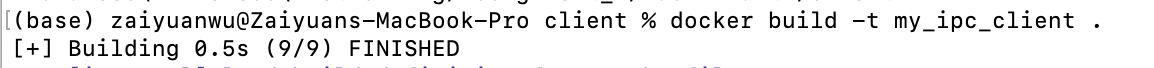
* Emulate another offloading scenario that is relevant for future applications.

Another scenario we implemented in this task in asymmetrical encryption with rsa-key pair. In the Part2 of this task, we emulated a client-server transmission, including the following steps:

* 1. server generate an rsa-key pair, sending public key to client
  2. client receive public key, then encrypt a secret message with it
  3. client send encrypted message to server
  4. server decrypt message with private key

Presentation:

1. Similar to last part, build two images:



1. Text

   Description automatically generatedRun containers separately:

Graphical user interface, text, application, email

Description automatically generated